

Fig. 1

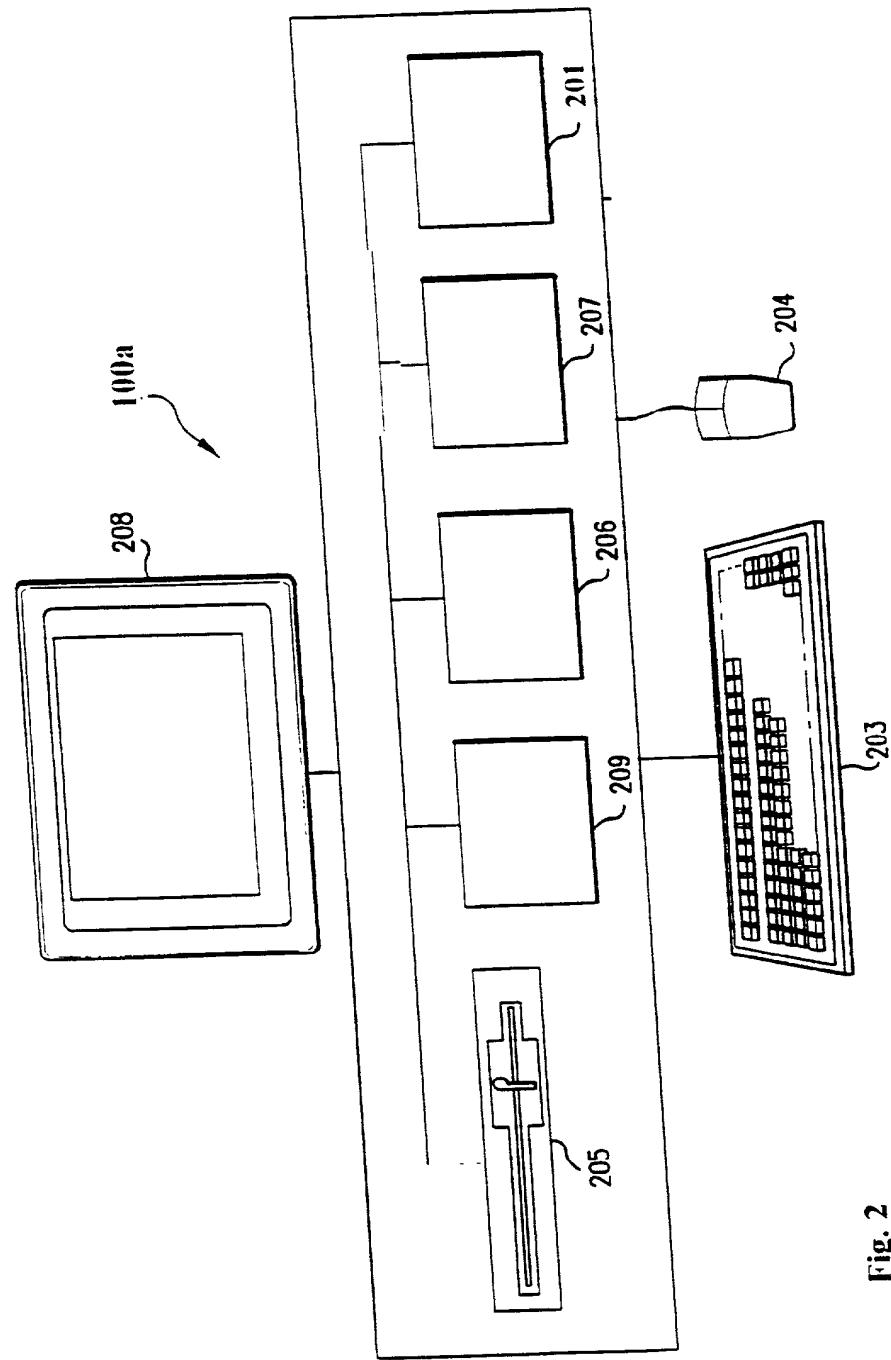


Fig. 2

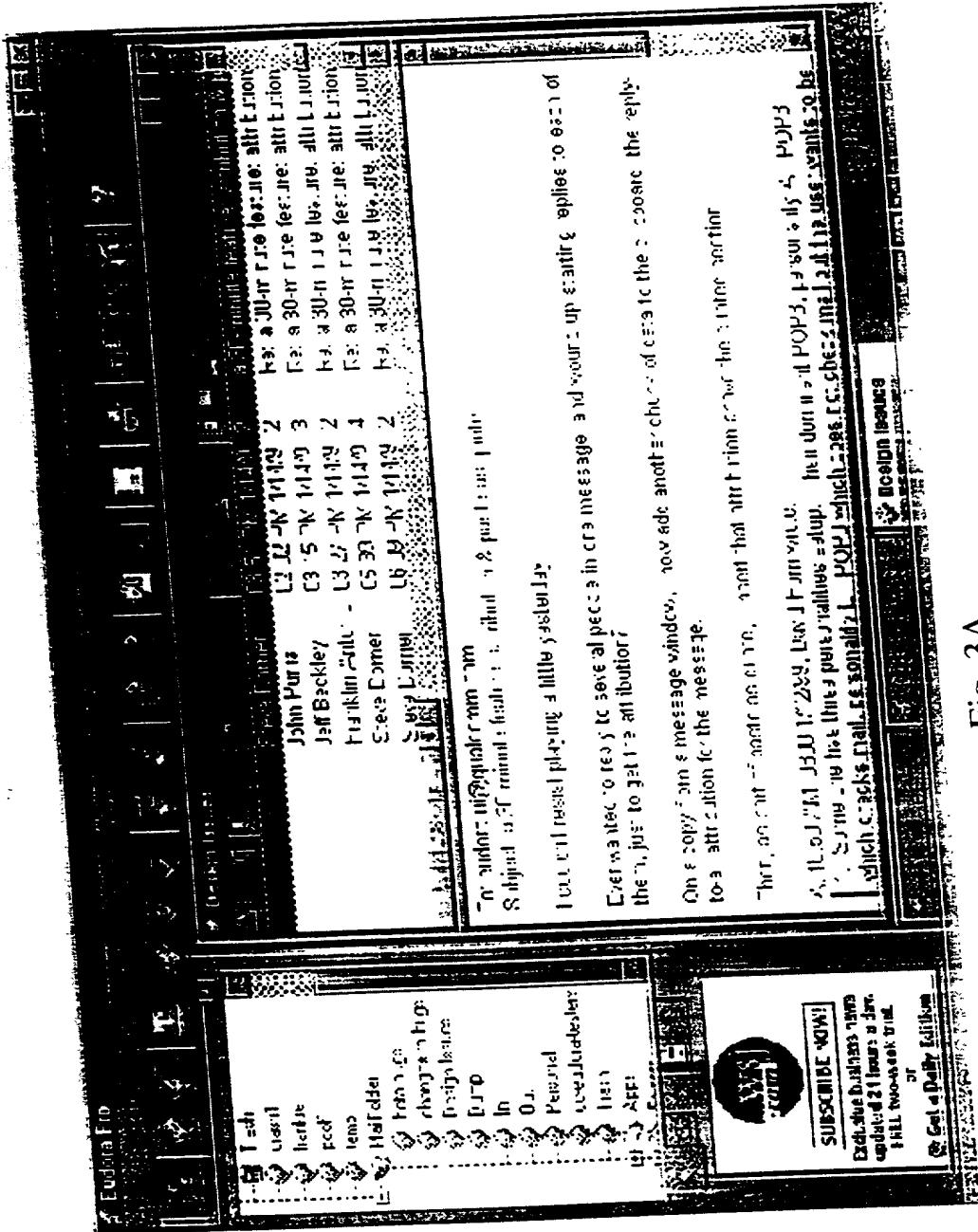


Fig. 3A

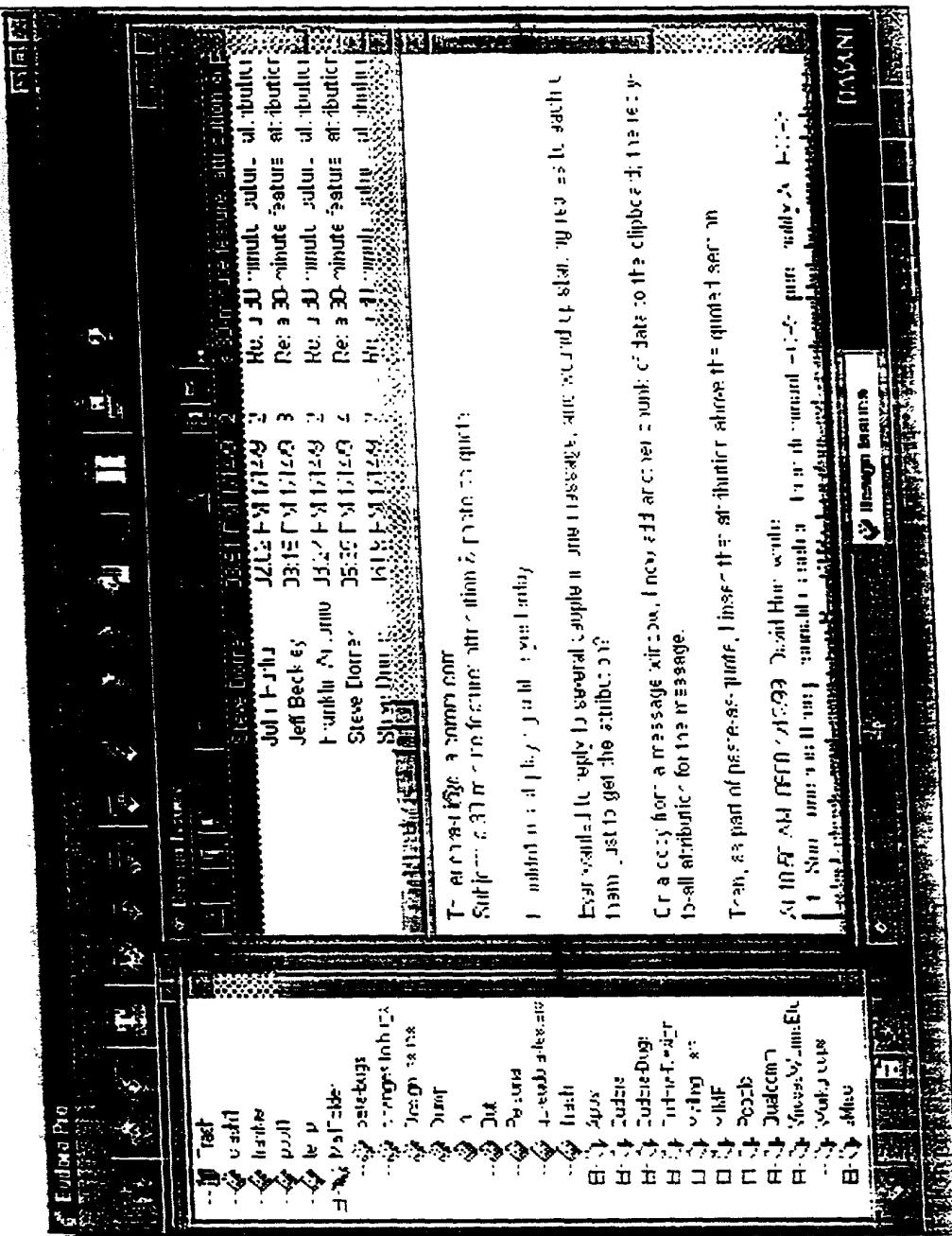


Fig. 3B

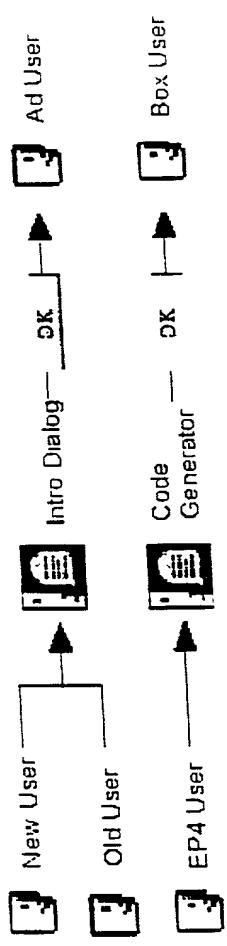


Fig. 4A

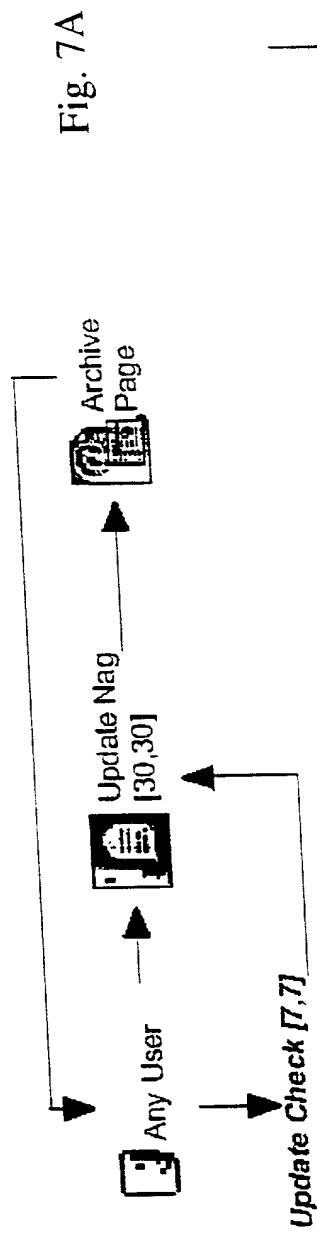


Fig. 7A

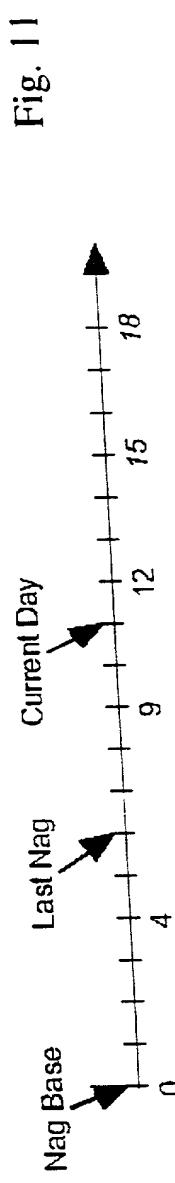


Fig. 11

**Welcome to Eudora!**  
Eudora is now licensed in three ways; Sponsored Mode, Paid Mode, and Light Mode. Unless you change modes, Eudora will run in Sponsored Mode, meaning it will display ads.

We have done our best to present the ads in a way that respects the work you do in email. By allowing Eudora to display ads, you get the full power of Eudora for free and we can still pay our bills.

Eudora shows Paid Mode features. If you decide the ads are not for you, you can change modes. Paid Mode for no ads. Current Eudora Pro 4.x users will be able to upgrade to Paid Mode. At this stage in testing, the machinery for Paid Mode is not fully tested, and Paid Mode is unavailable. Light Mode also shows no ads, but has many fewer features.

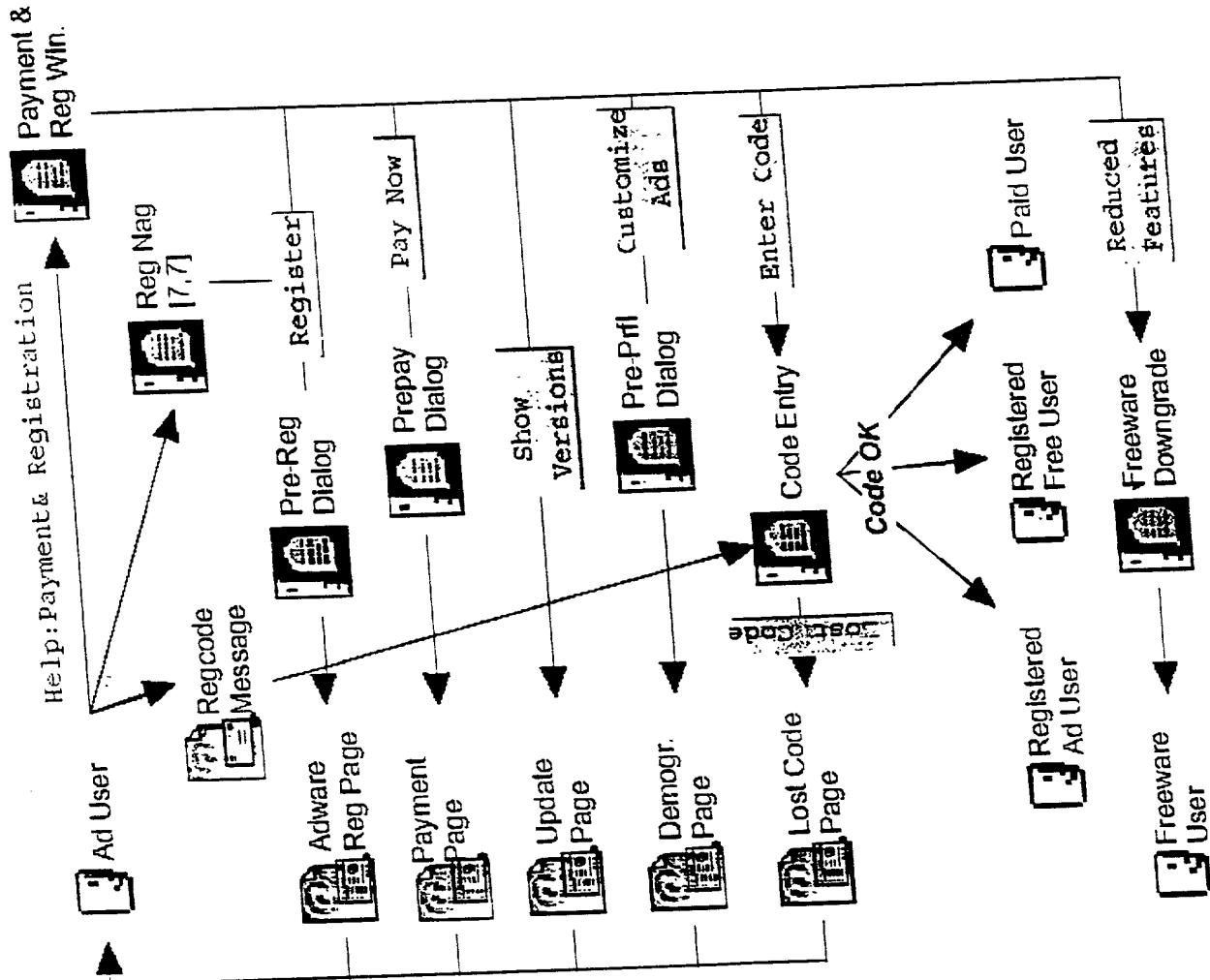
To switch forms of Eudora, please use the "Payment & Registration" item in the Help menu. To learn more about the three modes, click on the "Tell Me More" button below.

[Tell me more](#)

[OK](#)

Fig. 4B

Fig. 5A



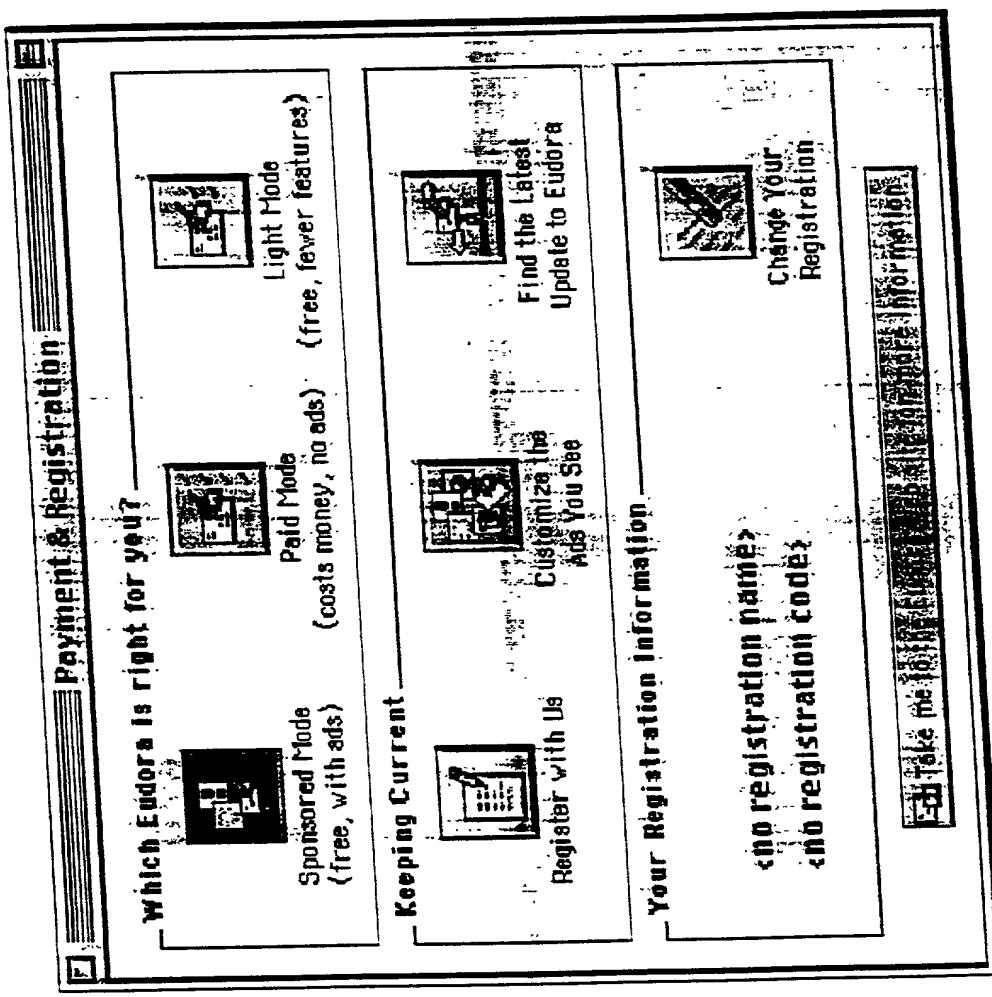


Fig. 5B

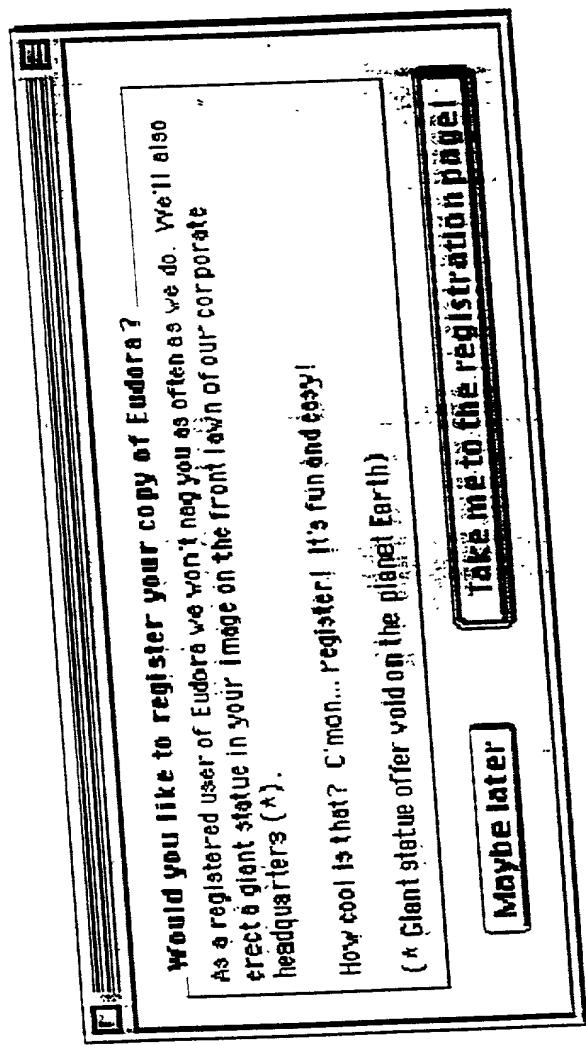


Fig. 5C

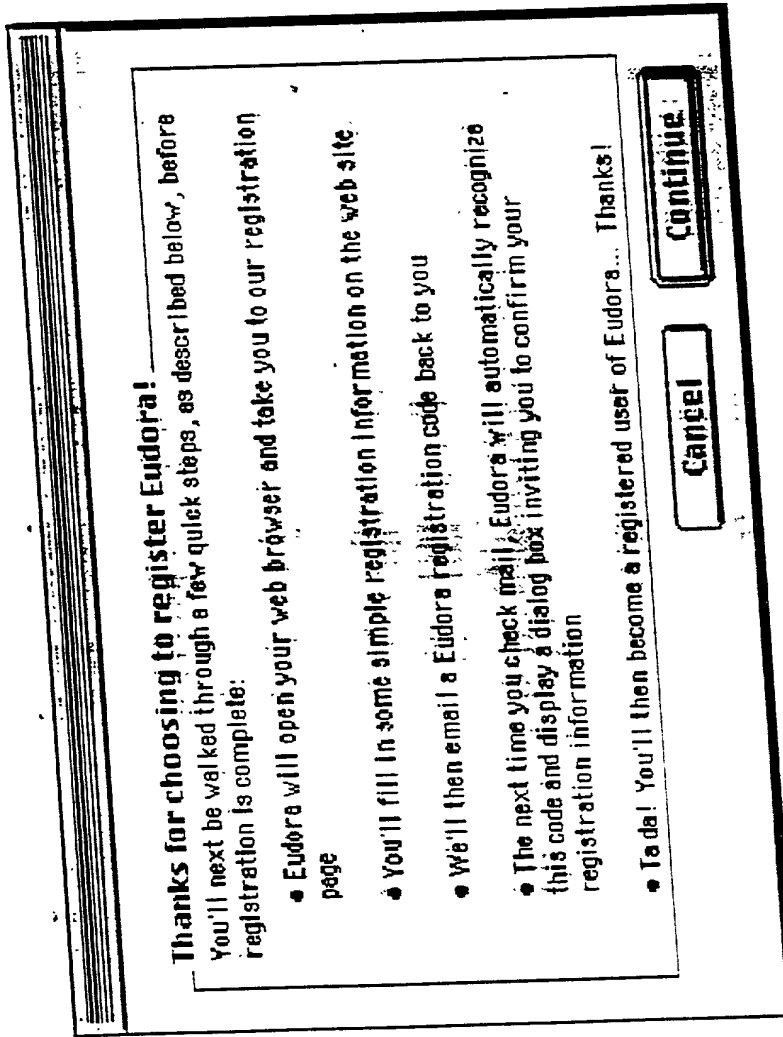


Fig. 5D

**Thanks for choosing to purchase Eudora!**

You'll next be walked through a few quick steps, as described below, before your purchase is complete:

- Eudora will open your web browser and take you to our Payment & Registration page
- You'll be asked to provide your payment and registration information on the web site
- You'll be asked to provide your payment and registration code back to you
- We'll then email a Eudora registration code back to you
- The next time you check mail, Eudora will automatically recognize this code and display a dialog box asking you to confirm your registration information
- Ta-dah! You'll then become a Paid Home user! Congratulations!

<input type="button" value="Cancel"/>	<input type="button" value="Continue"/>
---------------------------------------	---

**Fig. 5E**

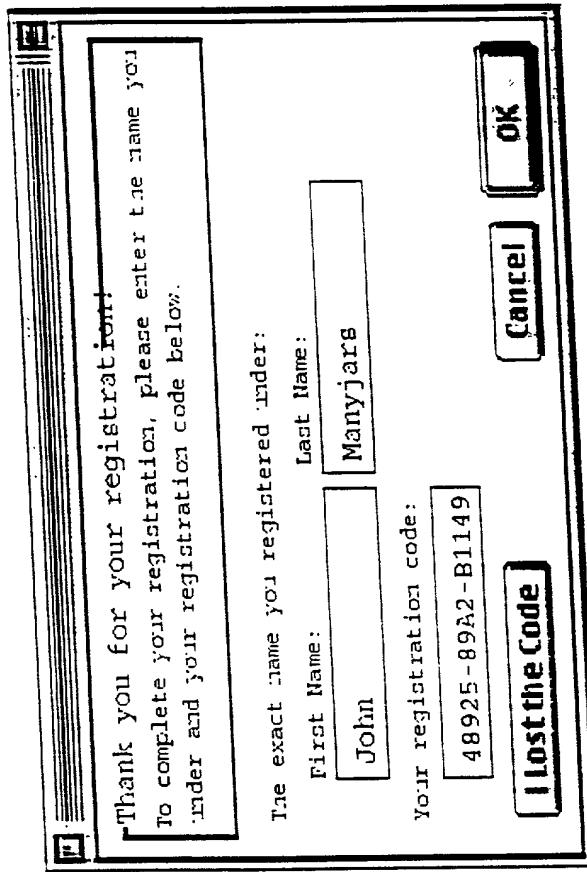


Fig. 5F

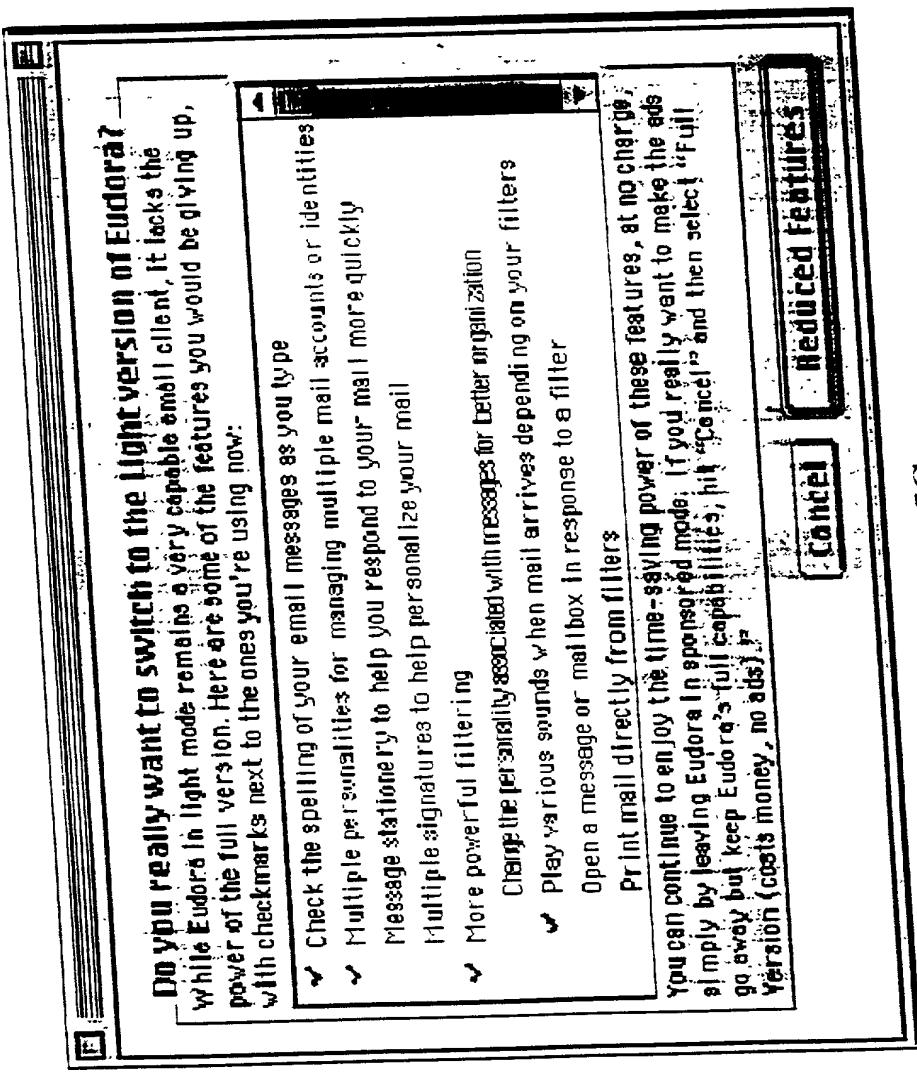
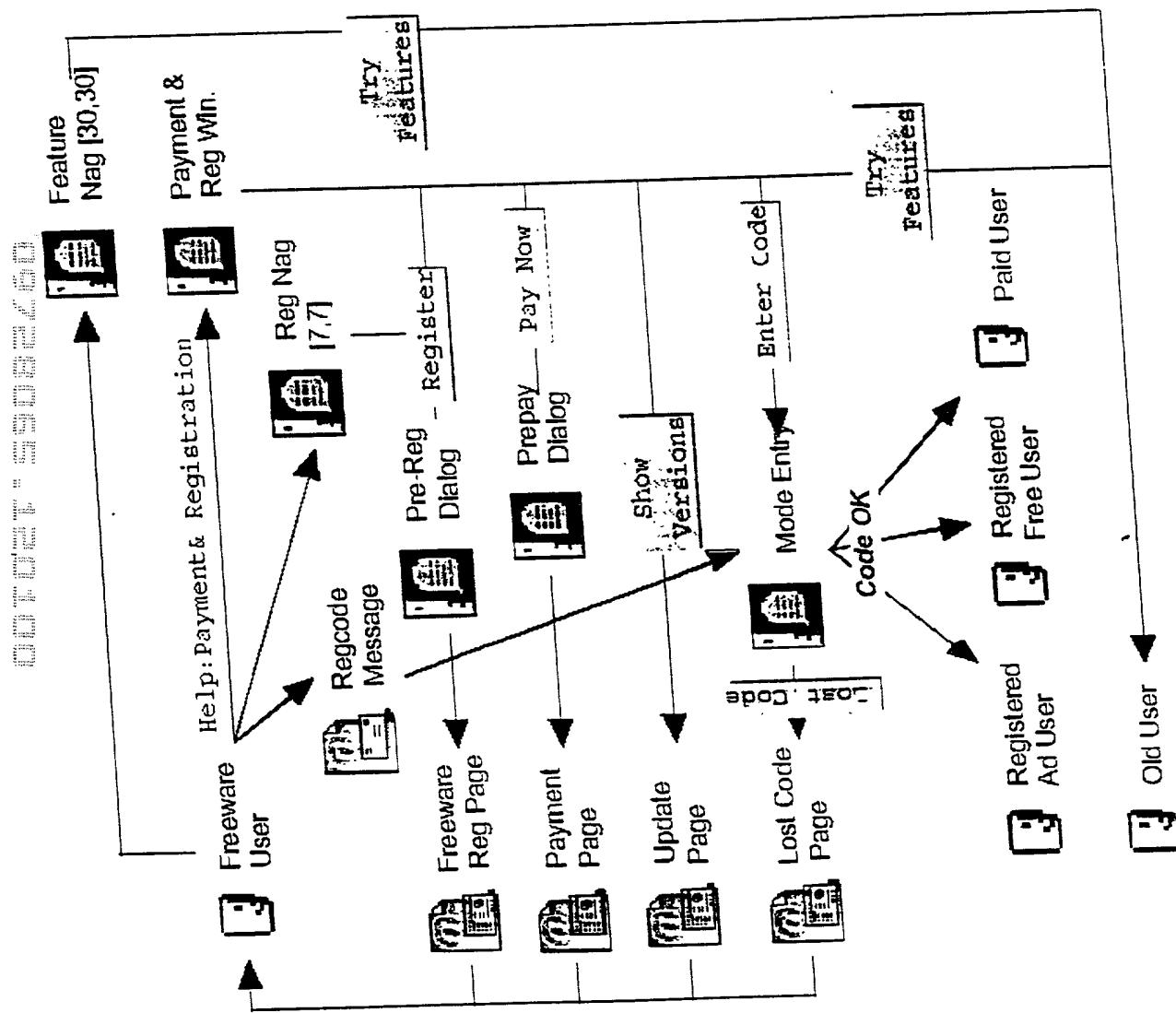


Fig. 5G

**Fig. 6A**



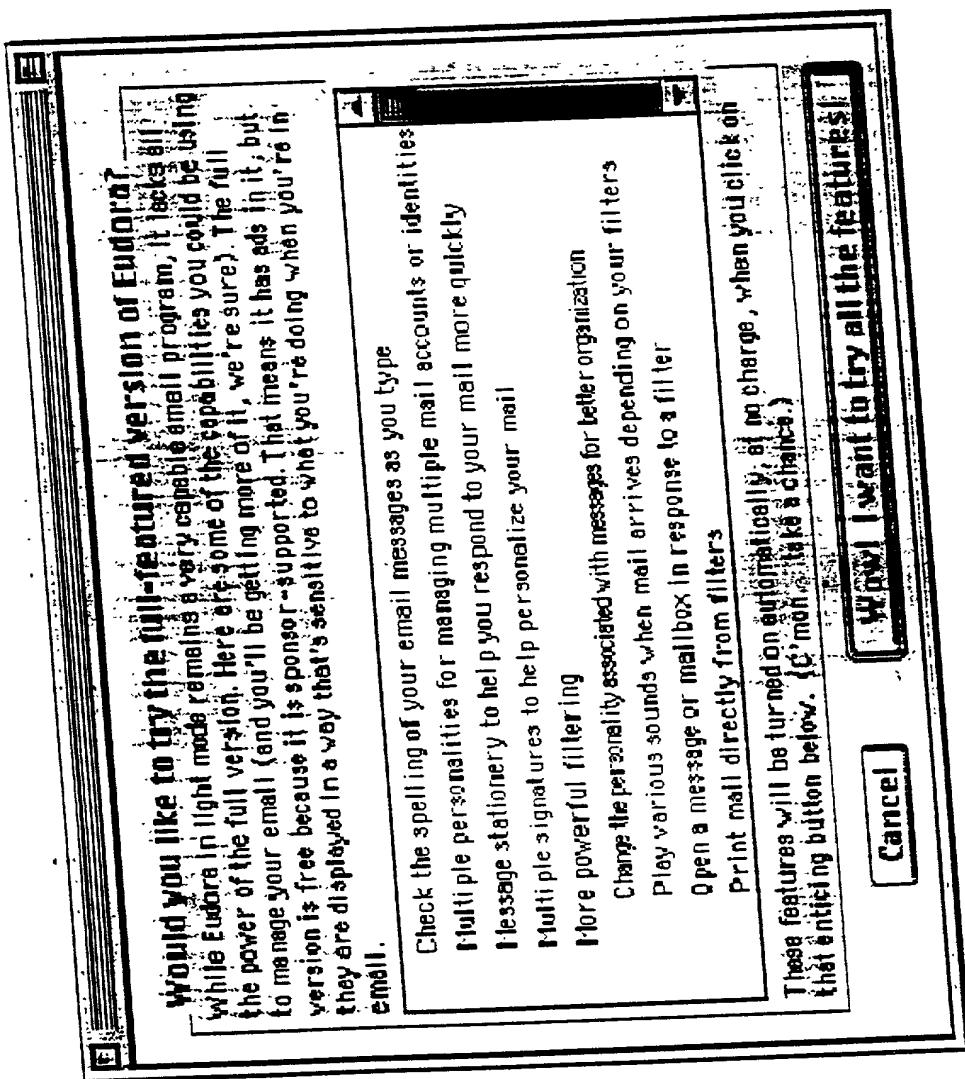


Fig. 6B

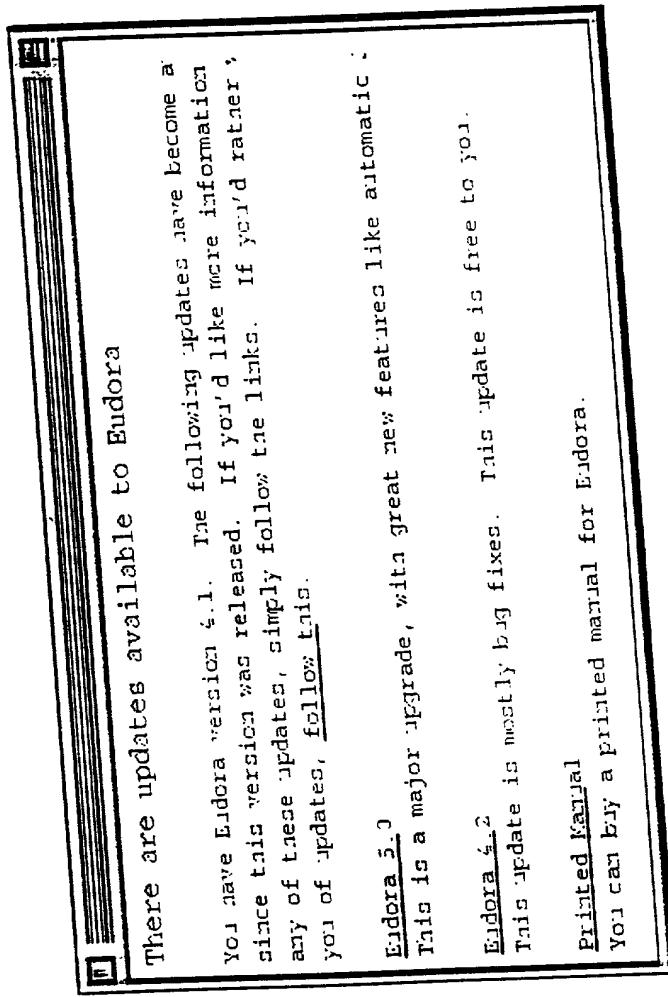
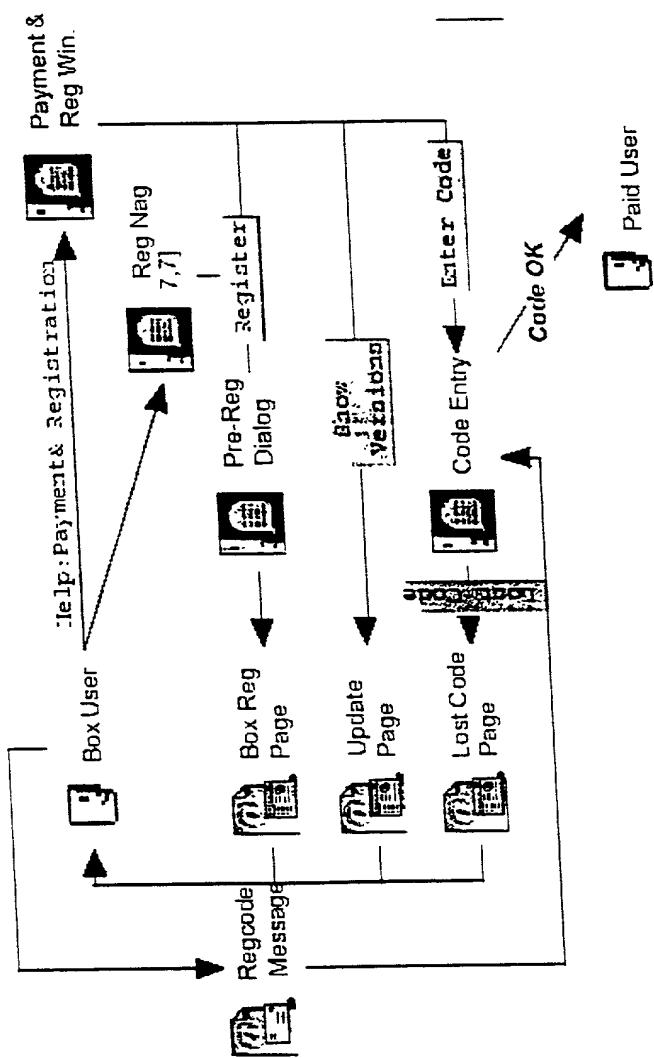


Fig. 7B

**Fig. 8**



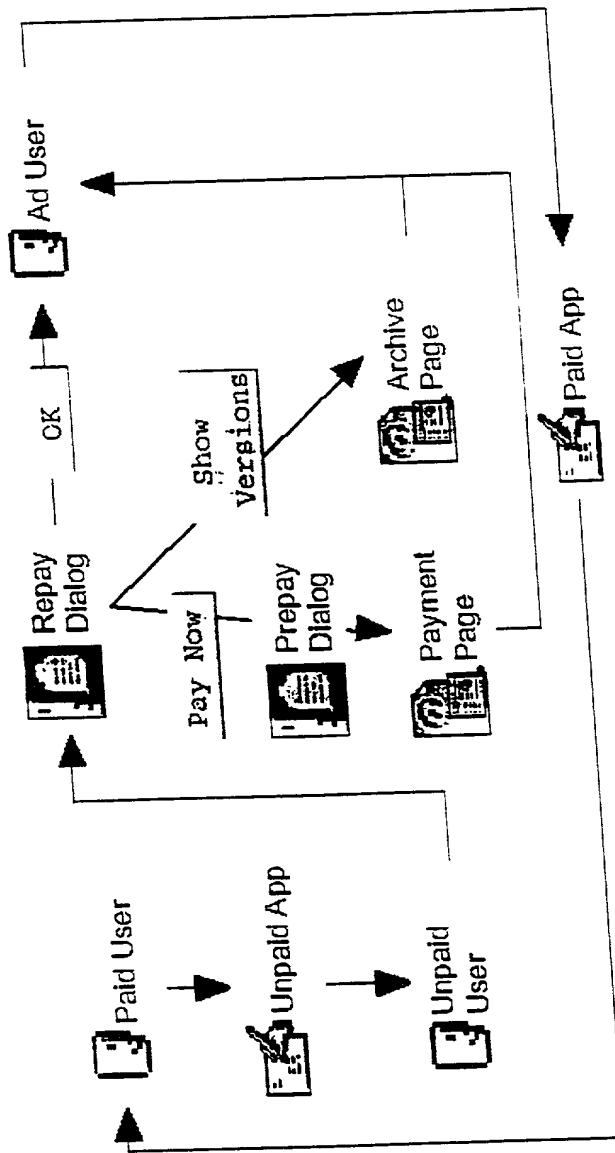


Fig. 9

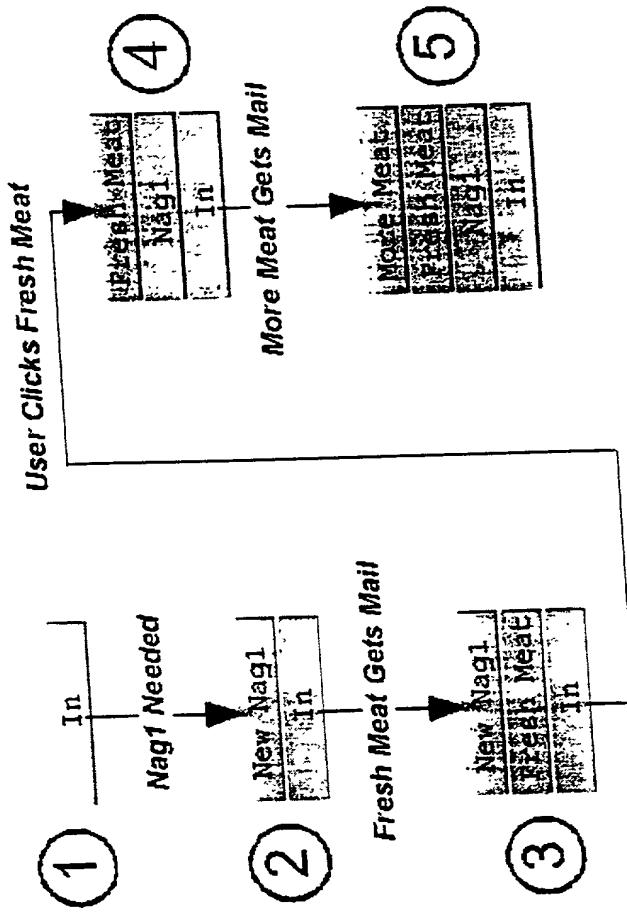


Fig. 10

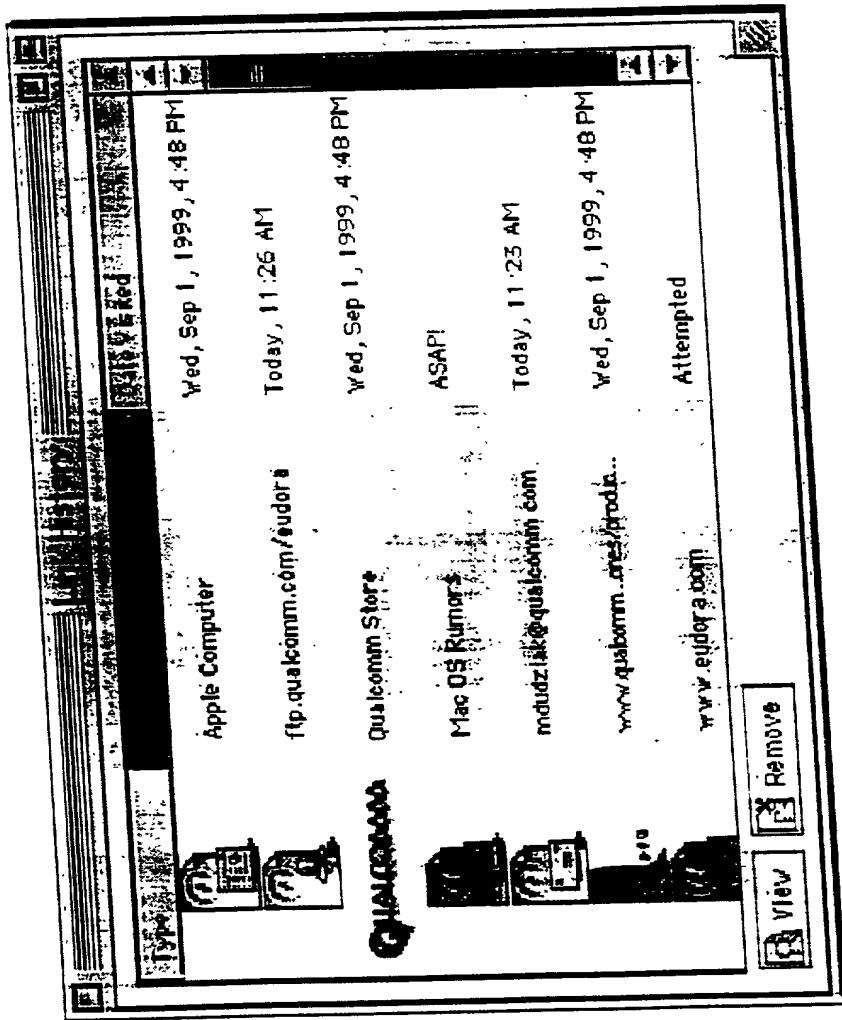


Fig. 12A

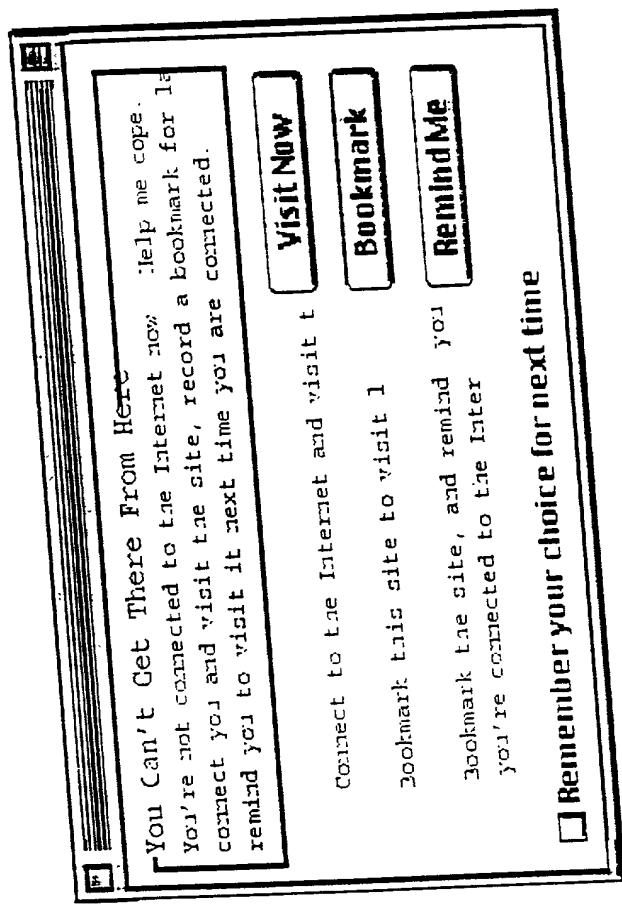


Fig. 12B

Plastic	Conc. Spec.	Size, in.	Plastic	Conc. Spec.	Size, in.	Plastic	Conc. Spec.	Size, in.	Plastic	Conc. Spec.	Size, in.
2.5 L.A.N.	C. min. - Spec'd.	2.5	2.5	9.1	9.1	3.000	3.000	2	2	5.000	5.000
Plastic	2.5 Size,	Plastic	Plastic	2.5	2.5	Plastic	Plastic	2	2	Plastic	Plastic
Plastic	2.5 Conc.	Plastic	Plastic	2.5	2.5	Plastic	Plastic	2	2	Plastic	Plastic
Plastic	2.5 Spec'd.	Plastic	Plastic	2.5	2.5	Plastic	Plastic	2	2	Plastic	Plastic

Fig. 13A

Influenza		Sv. Gute		Flu, Lie-	
Ex. Gute	Ex. Flu	Ex. Gute	Ex. Flu	Ex. Gute	Ex. Flu
451 No.	26	10	101	13	36
Age Per 100 Students & Staff	39	12	135	17	48
Age Per Person	52	12	135	17	60
Days	65	19	202	25	72
Days	78	19	202	25	72
Days	90	23	235	29	84
Days	35	90	235	35	12

Fig. 13B

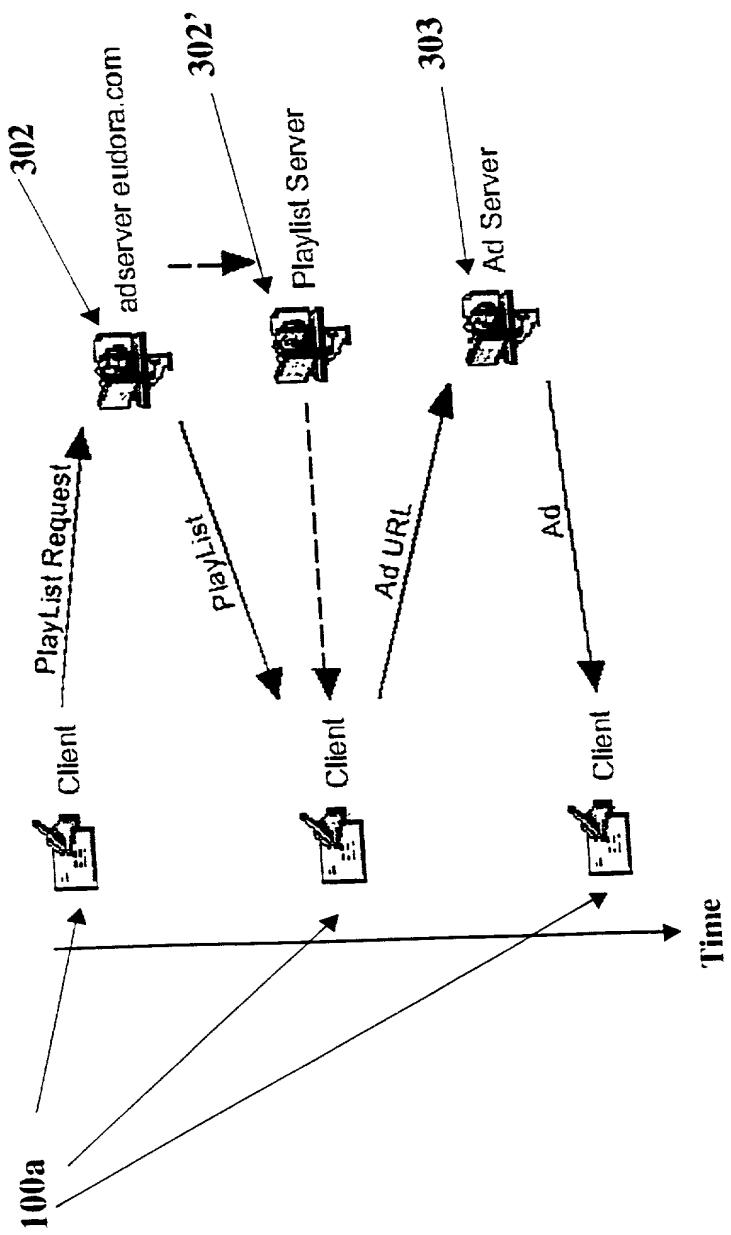


Fig. 14

```
//////////  
// Main ad scheduler  
ScheduleMain  
{  
    // Has a new day dawned?  
    Do CheckForNewDay  
    // Are we are within the current ad's showFor?  
    if ( ad.thisShowTime < ad.showFor )  
    {  
        // there is nothing to be done  
        return  
    }  
    // At this point, we know that we need a new ad  
    // Perform housekeeping tasks on the old one  
    Do AdEndBookkeeping  
    // Pop out of a block if all ads on par  
    if ( block isn't all playlists )  
    {  
        find ad with minimum ad.numberShown  
        if ( ad.numberShown >= blockGoal )  
            set block to all playlists  
    }  
    // If we are over our quota of regular ads for the day,  
    // look for a runout  
    if ( adFaceTimeToday > faceTimeQuota )  
    {  
        Do ShowARunout  
    }  
    else  
    {  
        Do ShowARegularAd  
    }  
}  
// end ad schedule main
```

Fig. 15A

CODE: ADVERTISING

```
///////////
// We must perform certain tasks when the calendar day
changes.
CheckForNewDay
{if ( the calendar day has changed )
{
// Perform housekeeping tasks on the ad currently showing
Do StopShowingCurrentAd
// Runout ads are charged for a full showFor if they've been
shown
// at all on a given day. Charge any runout ads if they've
been
// shown at all.
for runout ads
{
if ( ad.thisShowTime > 0 )
{
ad.totalTimeShown += ad.showFor
ad.thisShowTime = 0
}
}
// Now, reset the counters for all ads to reflect the fact
that
// a new day has dawned.
for all ads
{
ad.numberShownToday = 0
}
// Record yesterday's facetime
// Might not literally be yesterday, be sure to use
// whatever day the app was last run on
set old current day's facetime to totalFaceTimeToday
// and reset our global regular ad facetime counter
adFaceTimeToday = 0
totalFaceTimeToday = 0
// if we were in a block, back out
set block to all playlists
}
}
// end CheckForNewDay
```

Fig. 15B

```
///////////
// This function shows a runout ad, and if it
// can't find one, goes to a rerun
ShowARunout
{
for runout ads
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this runout today?
if ( ad.numberShownToday > ad.dayMax )
try next ad // this one's used up for the day
// are we done showing this runout for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next runout ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next runout ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this runout
// we are now in runout state
Do ShowAnAd
return
}
// if we haven't found a runout ad, we will go to "rerun"
state
Do ShowARerun
}
// end ShowARunout
```

Fig. 15C

```
///////////
// Rerun state. Look for a regular ad to rerun
ShowARerun
{
for regular ads [ in current block ]
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// is this ad recent enough to rerun?
if ( ad.lastShownDate is older than returnInterval )
try next ad
// this one is too old to rerun
// if in block, show ads only if it's their "turn"
if ( ad.numberShownToday >= blockGoal )
try next ad // need to find a friend in this block
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, at this point we can show this ad, but because
// we're in rerun, we don't keep the books
Do ShowAnAd
return
}
// if we get here, we have no ads to show. Punt.
return
}
// end ShowARerun
```

Fig. 15D

```
///////////
// Show a regular ad
ShowARegularAd
{
for regular ads [ in current block ]
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this ad today?
if ( ad.numberShownToday > ad.dayMax )
try next ad // this one's used up for the day
// if in block, show ads only if it's their "turn"
if ( ad.numberShownToday >= blockGoal )
try next ad // need to find a friend in this block
// are we done showing this ad for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this ad
// we are now in regular state
Do ShowAnAd
return
}
// If we get here, we have failed to find a regular
// ad. Go to runout
Do ShowARunout
}
// end ShowARegularAd
```

Fig. 15E

```
///////////
// Perform necessary housekeeping when we're taking
// down an ad
AdEndBookkeeping
{
    // In rerun state, we don't do any bookkeeping
    if ( in RerunState )
        return
    // Account for at most ad.showFor seconds, provided
    // we've shown the ad for at least ad.showFor seconds
    // Note that this means we don't charge for time beyond
    // ad.showFor seconds, which is important
    if ( ad.thisShowTime >= ad.showFor )
    {
        ad.numberShownToday += ad.showFor
        ad.showFor++
        // we do NOT reset thisShowTime here, we do it in
        // AdStartBookkeeping. It actually doesn't matter where
        // we do it, provided we are careful NOT to do it for
        // runout ads.
    }
}
// end AdEndBookkeeping
```

Fig. 15F

```
///////////
// Show an ad, including bookkeeping and block handling
ShowAnAd
{
    // If the ad is in a block, notice that
    if ( it's in a "block" playlist )
    {
        if ( not currently in a block )
        {
            find ad in block with minimum numberShown
            make that our ad
            set blockGoal to minimum numberShown+1
        }
        set current block to this playlist
    }
    // now do bookkeeping
    Do AdStartBookkeeping
    // and actually show it
    Do DisplayThatAd
}
```

Fig. 15G

```
///////////
// Perform housekeeping when we put up an ad
AdStartBookkeeping
{
// In rerun state, we don't do any bookkeeping
if ( in RerunState )
return
// For regular ads
if ( it's a regular ad )
{
ad.thisShowTime = 0
ad.lastShownDate = now
}
}
// end AdStartBookkeeping
```

<b>Persistent Ads</b>		
<b>PlayList Request</b>	<code>faceTime</code>	Used to determine how much advertising to send to client
	<code>faceTimeLeft</code>	Not used
<b>PlayList Response ClientInfo</b>	<code>reqInterval</code>	Relatively large: one or more days
	<code>flush</code>	Used. Single playlist completely specifies list of ads client should have
<b>PlayList Response Scheduling Parameters</b>	<code>showForMax</code>	Not used
<b>PlayList Response Scheduling Parameters</b>	<code>showForMax</code>	Not used

**Fig. 16A**

<b>Short-Lived Ads</b>		
<b>PlayList Request</b>	<code>faceTime</code>	Not used
	<code>faceTimeLeft</code>	Used to determine how many ads client should receive
<b>PlayList Response ClientInfo</b>	<code>reqInterval</code>	Not used. Instead client requests new playlist whenever ads "run low"
	<code>flush</code>	Not used
<b>PlayList Response Scheduling Parameters</b>	<code>showForMax</code>	Used to determine how long an ad runs

**Fig. 16B**

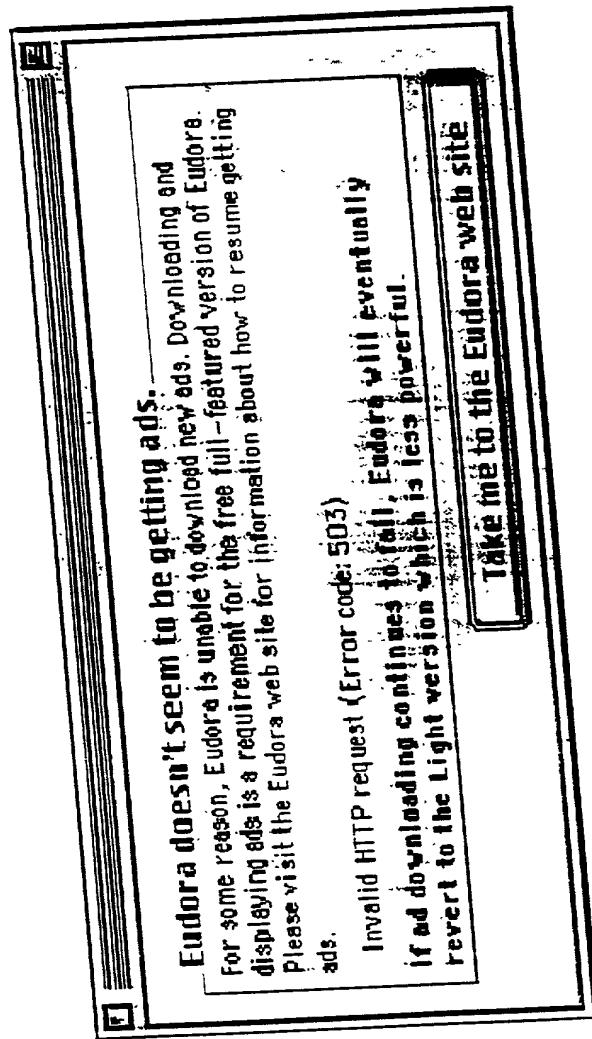


Fig. 17A

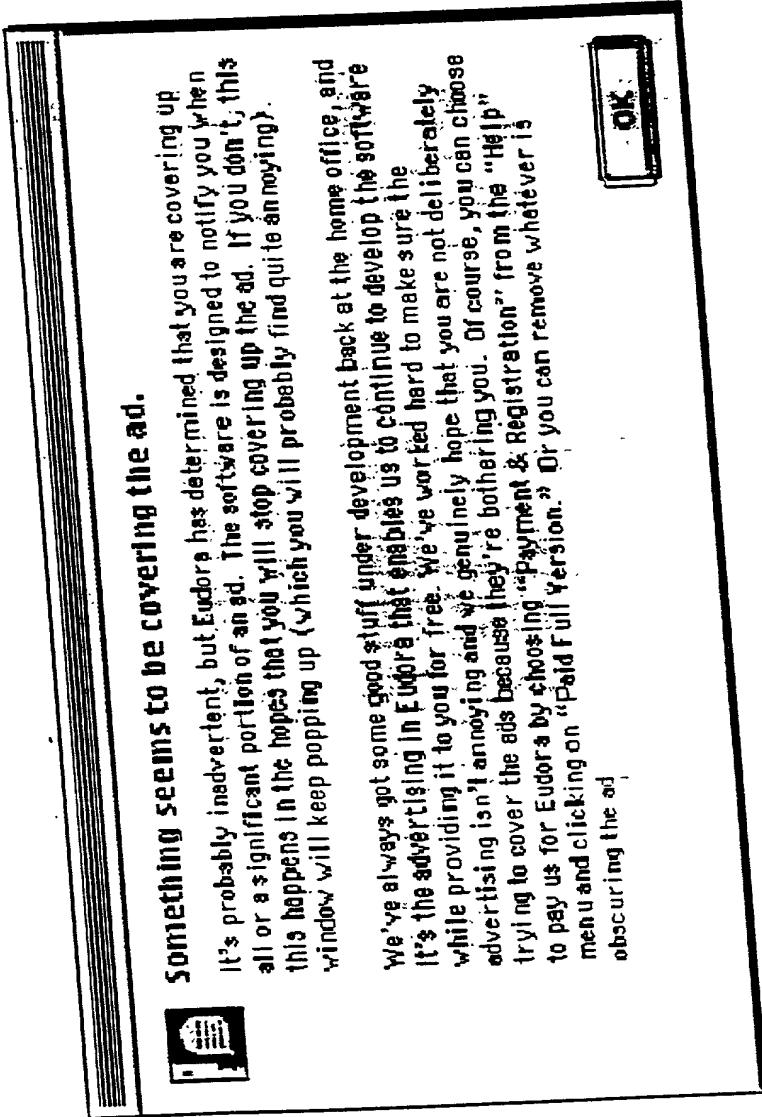


Fig. 17B

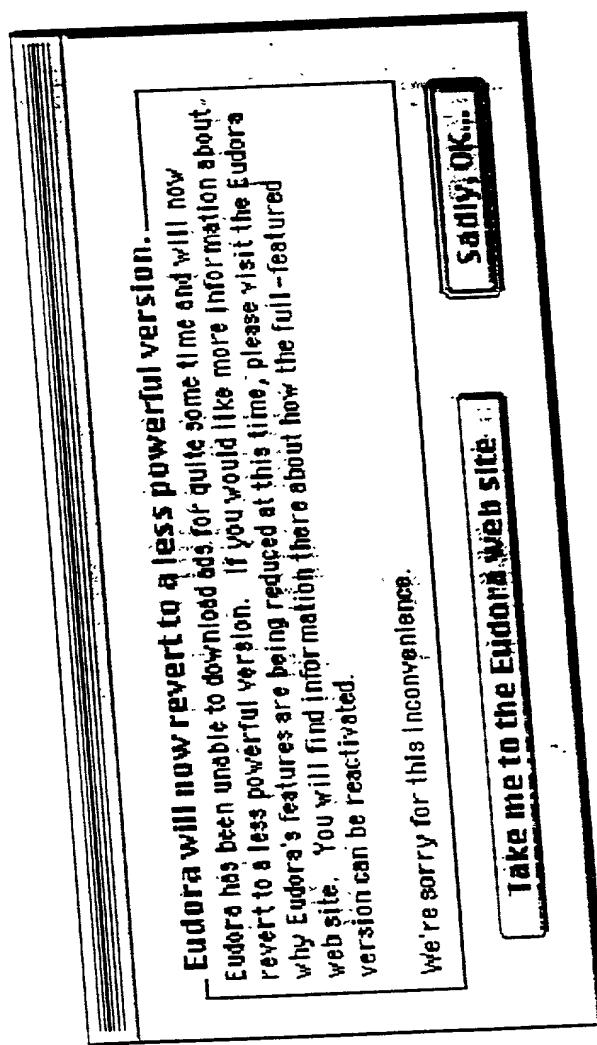


Fig. 17C

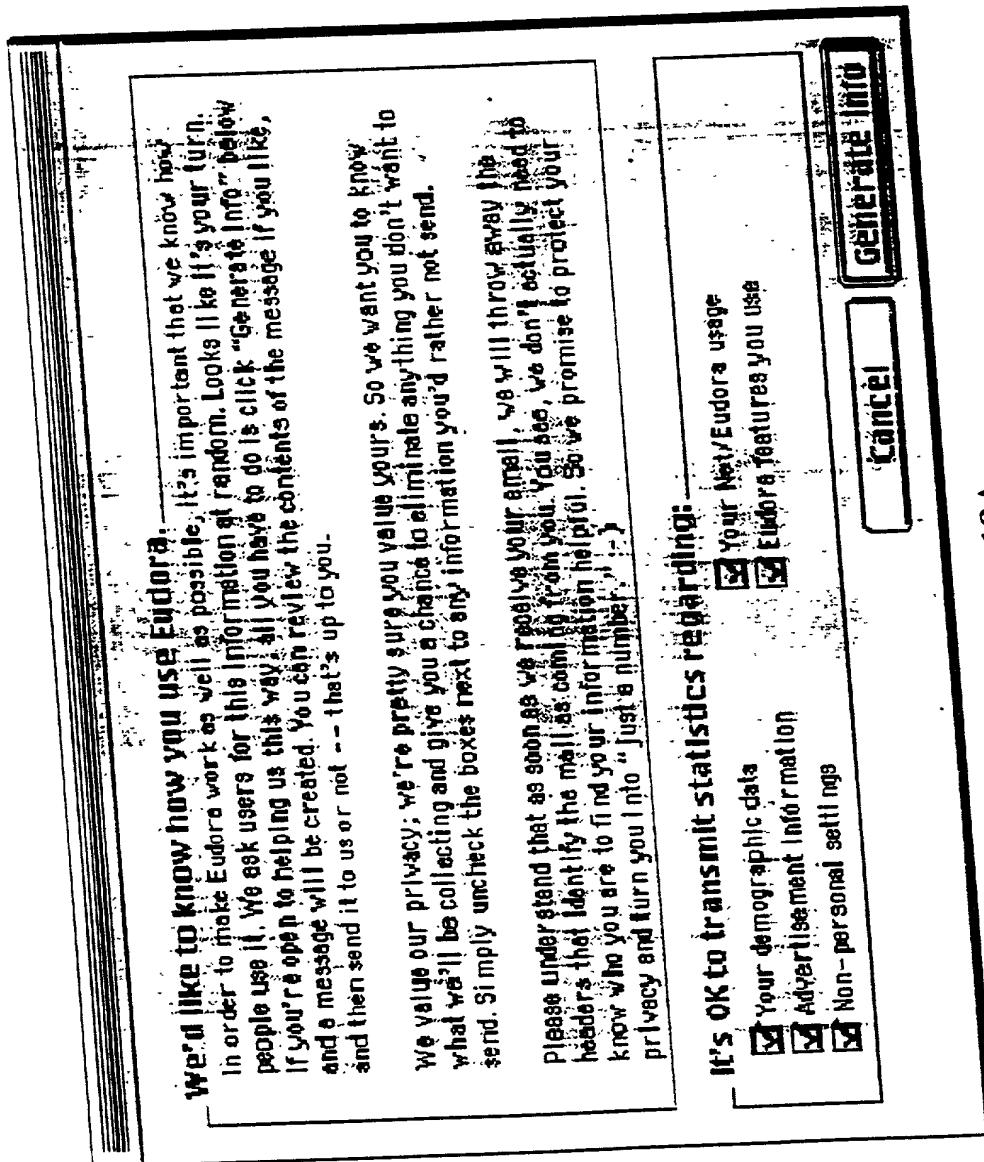


Fig. 18A

**Fig. 19**

Page		Applicable Query Parts														
		action	platform	product	version	distributor	mode	email	firstname	regfirst	reglast	regcode	oldReg	profile	uri	adid
Payment	pay	X	X	X	X	X	X	X	X	X	X	X	X			
Freeware Registration	register-free	X	X	X	X	X	X	X	X	X	X	X	X			
Adware Registrations	register-ad	X	X	X	X	X	X	X	X	X	X	X	X			
Box Registrations	register-box	X	X	X	X	X	X	X	X	X	X	X	X			
Lost Code	lostcode	X	X	X	X	X	X	X	X	X	X	X	X			
Update	update	X	X	X	X	X	X	X	X	X	X	X	X			
Pro Update	proupdate	X	X	X	X	X	X	X	X	X	X	X	X			
Archived	archived	X	X	X	X	X	X	X	X	X	X	X	X			
Profile	profile	X	X	X	X	X	X	X	X	X	X	X	X			
Introduction	intro	X	X	X	X	X	X	X	X	X	X	X	X			
Support	n/a	X	X	X	X	X	X	X	X	X	X	X	X			
QuickTime Missing	support	X	X	X	X	X	X	X	X	X	X	X	X			
Ad Failure	support	X	X	X	X	X	X	X	X	X	X	X	X			
Tutorial	support	X	X	X	X	X	X	X	X	X	X	X	X			
FAQ	support	X	X	X	X	X	X	X	X	X	X	X	X			
Light Users	support	X	X	X	X	X	X	X	X	X	X	X	X			
Search Support	support	X	X	X	X	X	X	X	X	X	X	X	X			
Newsgroups	support	X	X	X	X	X	X	X	X	X	X	X	X			

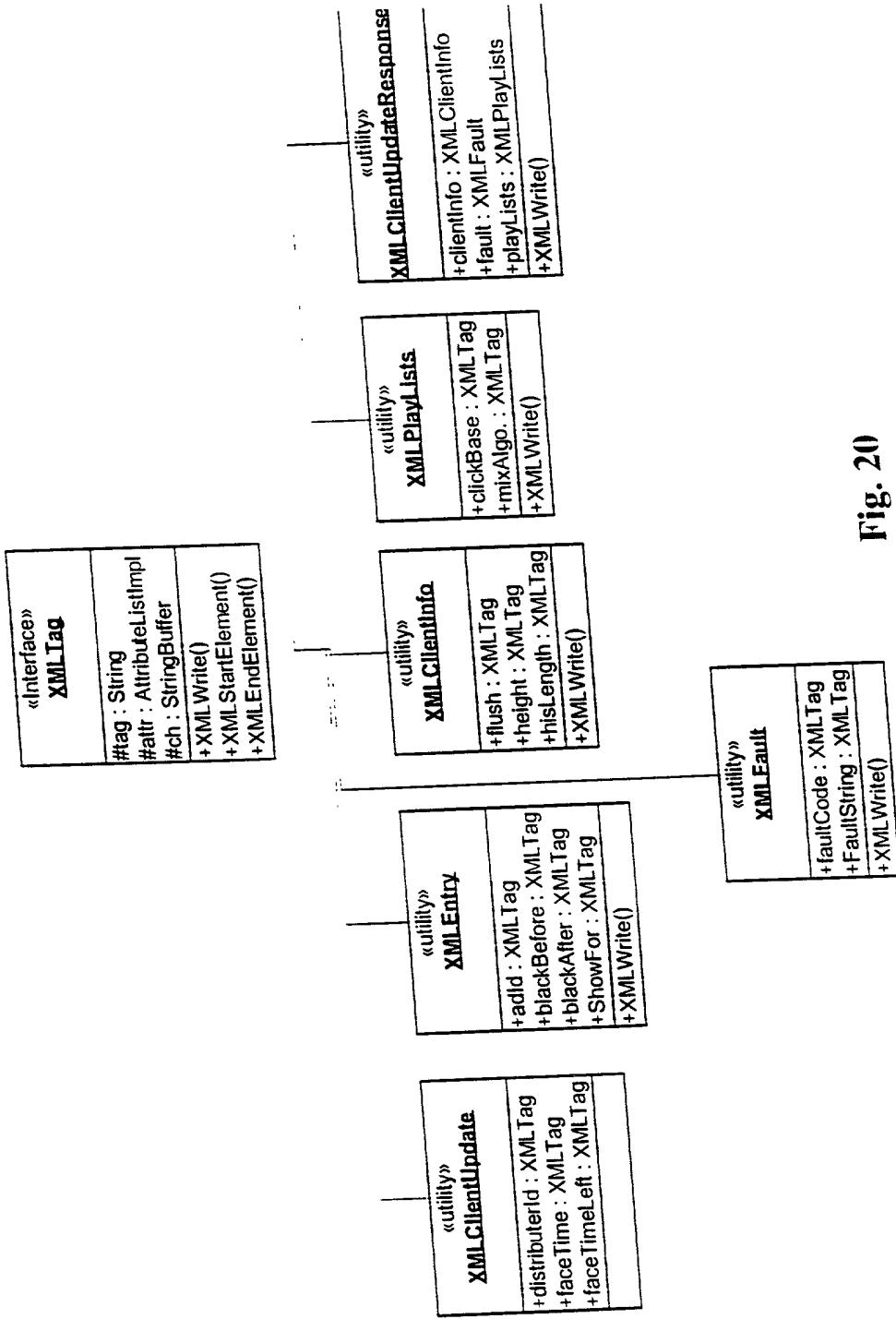


Fig. 20

§ The list of available ads advantageously can be built from the following query:

```
ads = dbCon.prepareStatement("SELECT * FROM ads WHERE startDate <= today AND endDate >= today + 30 AND AdType = 'P' AND AdStatus = 'A' AND ImpressionsServed < Impressions ORDER BY ImpressionsServed ASC");
run out ads = dbCon.prepareStatement("SELECT * FROM ads WHERE startDate <= today AND endDate >= today + 30 AND AdType = 'R' AND AdStatus = 'A' AND ImpressionsServed < Impressions ORDER BY ImpressionsServed ASC");
```

§ The time required to deliver the ads advantageously can be calculated in the following manner.

```
faceTimeLeftForToday = faceTime[today] - faceTime[seedOfDay]
```

faceTimeLeftForToday [seconds] = faceTime[today] – faceTime[seedOfDay]  
(Comment: Face time left for today is the number of seconds the servlet can use to deliver special ads today.)

```
predictFaceTimeLeftForTomorrow = sum(faceTime[tomorrow], faceTime[tomorrow + 1], ..., faceTime[tomorrow + reqInterval])  
predictFaceTimeLeftForTomorrow [seconds] = sum(faceTime[tomorrow], faceTime[tomorrow + 1], ..., faceTime[tomorrow + reqInterval])  
(Comment: Predict face time is the number of seconds the servlet predicts the user is going to have.)
```

(Comment: Predict face time is the number of seconds the servlet predicts the user is going to have.)  
goalShowTimeLeft [seconds] = predictFaceTime - faceTimeLeftForToday  
(Comment: Goal show time left is the number of seconds that the software provider needs to fill with ads.)

Fig. 21A

```

8 Targeting {
    while (faceTimeLeftForToday) {
        if ad is not in the history {
            select ad [according to target = today]
            faceTimeLeftForToday -= ad.showFor
        }
        next ad
    }

    while (goalShowTimeLeft) {
        if ad is not in the history {
            select ad [according to target]
            goalShowTimeLeft -= ad.showFor
        }
        next ad
    }
}

```

Default values:

`reqInterval = 1 day.`  
`faceTime = 30 minutes`  
`faceTimeQuota is ?`  
`histLength = 31 days`

Fig. 21B

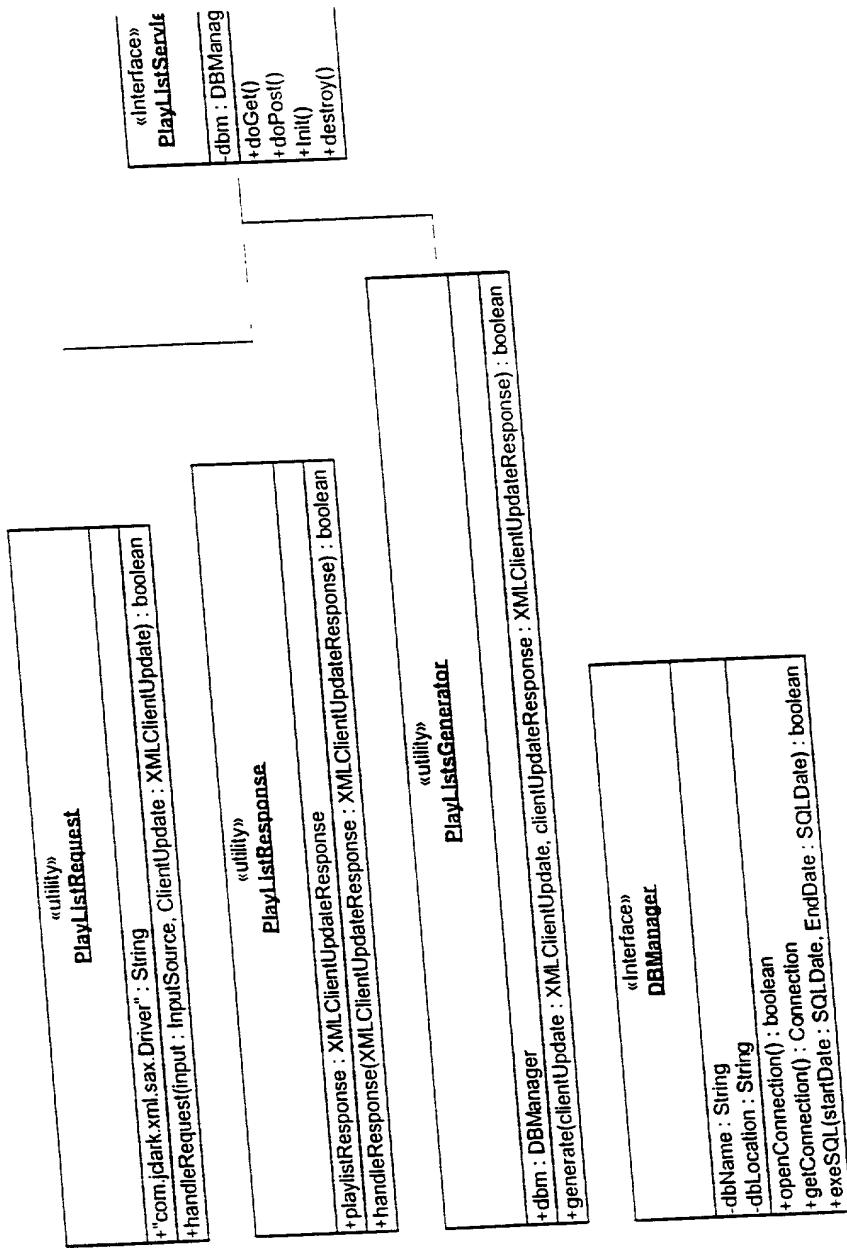
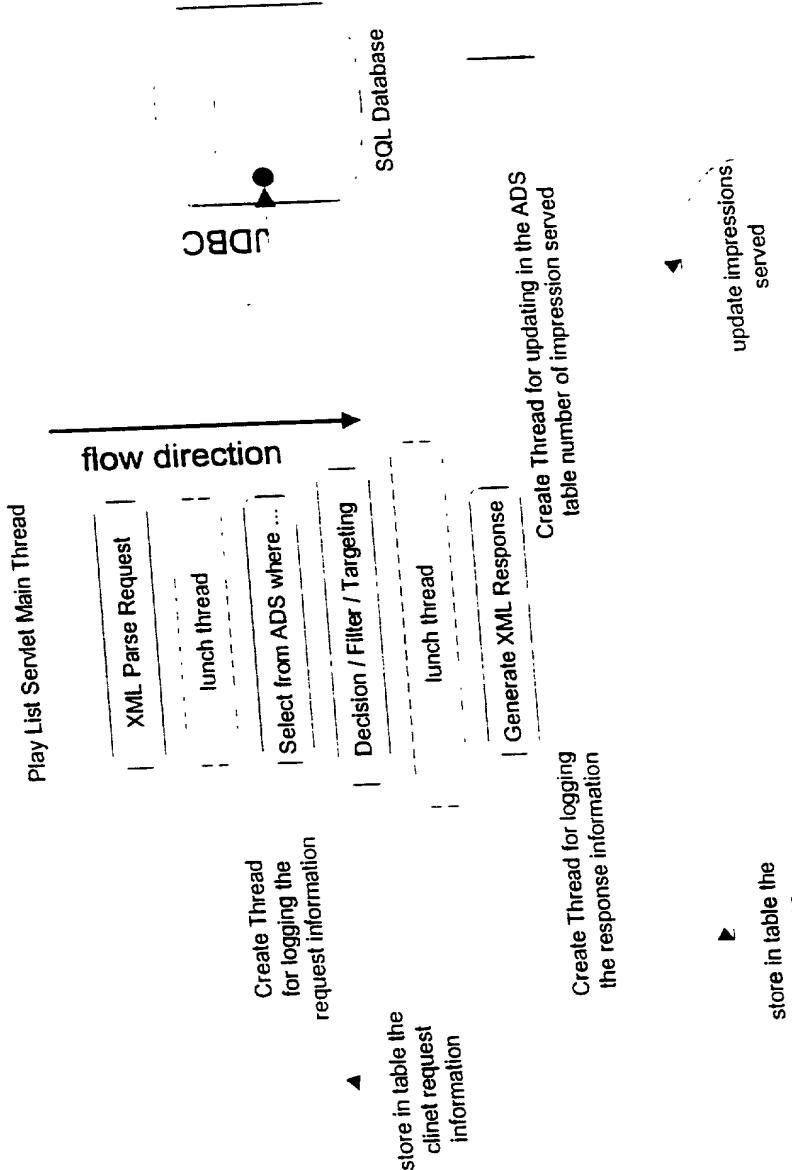


Fig. 22



**Fig. 23**